

AD-A158 169

AN ALGORITHM TO TRANSLATE RELATIONAL ALGEBRA QUERIES
INTO QUEL(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON
AFB OH D F BLUMENTHAL MAY 85 AFIT/CI/NR-85-57T

1/1

UNCLASSIFIED

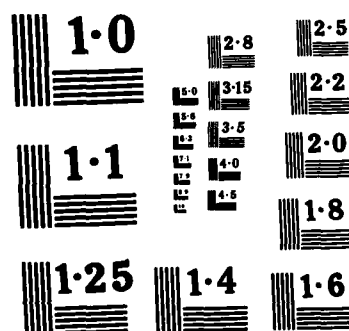
F/G 9/2

NL

END

FILMED

100



NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

AD-A158 169

1

AN ALGORITHM TO TRANSLATE RELATIONAL
ALGEBRA QUERIES INTO QUEL

BY

DENNIS F. BLUMENTHAL

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

1985

DTIC FILE COPY

This document has been approved
for public release and sale; its
distribution is unlimited.

2
S
AUG 19 1985
1

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER AFIT/CI/NR 85-57T	2. GOVT ACCESSION NO. AD-A158169	3. REPORT'S CATALOG NUMBER	
4. TITLE (and Subtitle) An Algorithm to Translate Relational Algebra Queries into QUEL		5. TYPE OF REPORT & PERIOD COVERED THESIS/DISSERTATION	
		6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s) Dennis F. Blumenthal		8. CONTRACT OR GRANT NUMBER(s)	
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: University of Florida		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433		12. REPORT DATE May 1985	
		13. NUMBER OF PAGES 73	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASS	
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)			
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17 5 AUG 1985 LYNN E. WOLAVER Dean for Research and Professional Development AFIT, Wright-Patterson AFB OH			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		Accession For NTIS GPO DTIC TAB Unannounced Justified	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) ATTACHED		By Dist Avail Dist Spec	

DD FORM 1473
1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS

85

8

13

10

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to ascertain the value and/or contribution of research accomplished by students or faculty of the Air Force Institute of Technology (AFIT). It would be greatly appreciated if you would complete the following questionnaire and return it to:

AFIT/NR
Wright-Patterson AFB OH 45433

RESEARCH TITLE: An Algorithm to Translate Relational Algebra Queries into QUEL

AUTHOR: Dennis F. Blumenthal

RESEARCH ASSESSMENT QUESTIONS:

1. Did this research contribute to a current Air Force project?

☐ a. YES

☐ b. NO

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not?

☐ a. YES

☐ b. NO

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency achieved/received by virtue of AFIT performing the research. Can you estimate what this research would have cost if it had been accomplished under contract or if it had been done in-house in terms of manpower and/or dollars?

☐ a. MAN-YEARS _____

☐ b. \$ _____

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3. above), what is your estimate of its significance?

☐ a. HIGHLY
SIGNIFICANT

☐ b. SIGNIFICANT

☐ c. SLIGHTLY
SIGNIFICANT

☐ d. OF NO
SIGNIFICANCE

5. AFIT welcomes any further comments you may have on the above questions, or any additional details concerning the current application, future potential, or other value of this research. Please use the bottom part of this questionnaire for your statement(s).

NAME _____

GRADE _____

POSITION _____

ORGANIZATION _____

LOCATION _____

STATEMENT(s): _____

FOLD DOWN ON OUTSIDE - SEAL WITH TAPE

AFTT/NR
WRIGHT-PATTERSON AFB OH 45433

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE, \$300



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 73236 WASHINGTON D.C.

POSTAGE WILL BE PAID BY ADDRESSEE

AFTT/ DAA

Wright-Patterson AFB OH 45433



FOLD IN

ACKNOWLEDGEMENTS

I am grateful to Dr. Stanley Su for the opportunity to participate in this effort and for his patience and encouragement. I wish to thank the U.S. Air Force for funding my graduate studies and the National Bureau of Standards (NBS) for grant 60NANB4D0017. I also wish to thank Mr. Mohamed Khatib for his work on the translation program. Special recognition goes to Susan, my dear wife, who gave me moral support throughout the master's program.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENTS	ii
ABSTRACT	vi
CHAPTER	
I. INTRODUCTION	1
II. RELATIONAL ALGEBRA	5
Projection	6
Selection	7
Union	8
Intersection	9
Difference	9
Product	10
Join	11
Natural Join	11
III. ALGEBRAIC OPERATIONS WITH QUEL	15
Projection	15
Selection	16
Union	17
Intersection	18
Difference	19
Product	20
Join	21
Natural Join	22
IV. TRANSLATION ALGORITHM	24
Tree Processing	25
Operation Mapping	26
Projection	27
Selection	28
Union	28
Intersection	29
Difference	29
Product	30
Join	31
Natural Join	31

CHAPTER	Page
Examples	32
Query 1	33
Query 2	34
Query 3	34
V. MINIMIZATION	37
Minimization Rules	37
Type I	38
Consecutive projections	38
Consecutive selections	39
Mixed projections and selections	39
Type II	40
Union	40
Intersection	40
Difference	41
Product	41
Join	41
Natural Join	42
Optimal Algorithm	42
Union	42
Intersection	43
Difference	44
Product	44
Join	44
Natural Join	48
Examples	49
Query 1	49
Query 2	50
Query 3	50
VI. IMPLEMENTATION	52
Query Packet Format	52
Tree Descriptors	53
Operation Descriptors	55
Attribute lists	55
Predicates	55
Relation tables	56
Tree Processing	56

CHAPTER	<u>Page</u>
Operation Mapping	58
Statement Generation	58
Attribute Tracking	59
Minimization	59
VII. CONCLUSION	62
APPENDIX	64
BIBLIOGRAPHY	70
BIOGRAPHICAL SKETCH	73

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

An Algorithm to Translate Relational
Algebra Queries into QUEL

by

Dennis F. Blumenthal

May 1985

Chairman: Dr. Stanley Y. W. Su
Major Department: Computer and Information Sciences

An algorithm for translating relational algebra query trees into QUEL is presented. The algorithm is used to process queries in a distributed, heterogeneous data base management system used to support computer integrated manufacturing. A node-by-node translation algorithm as well as an optimal algorithm which uses minimization techniques is presented. A general discussion about implementation is also included.

Michael J. Flanagan For
Chairman

CHAPTER I INTRODUCTION

Computer integrated manufacturing (CIM) is an important goal of current manufacturing research underway at the National Bureau of Standards (NBS). A general discussion of that work is found in [Sim82]. The objective of CIM is to automate all phases of manufacturing, ranging from engineering design and analysis, to parts machining and materials handling, and to integrate them into a system which also supports automated management functions, such as, process planning, shop scheduling, and inventory control. Achieving this goal will undoubtedly facilitate factory management, permit more even utilization of resources, and result in increased productivity and reduced operating costs. Comprehensive introductory material to batch manufacturing is found in selected articles published in [IEE83].

To accomplish CIM, it is essential to integrate and manage the large body of data which comprises the manufacturing database. This database consists of many types of data which cannot be easily included in a single database management system (DBMS). Current automated manufacturing systems utilize autonomous data management resources of

varying complexity, ranging from simple file management systems to sophisticated database management systems. These systems usually differ from each other, not only in their internal data representation, but more importantly in their underlying data models, data definition languages (DDL) and data manipulation languages (DML). Discussions about DBMS architectures for manufacturing systems are found in [Bee83] and [Mac83]. A discussion on the use of homogeneous and heterogeneous, distributed data management systems is found in [Bee83]. In [NBS85] a heterogeneous, distributed DBMS, called Integrated Manufacturing Data Administration System (IMDAS), is proposed for the Automated Manufacturing Research Facility (AMRF). A description of the AMRF is found in [Sim82], [Mac82], and [Mac83].

In order to share a heterogeneous database through a network, it must be represented through a global schema that incorporates logical units which can differ in local structure and content. The IMDAS architecture of [NBS85] introduces a three-view concept of distributed data and is shown in Figure 1. A global external view is a given component system's view of the distributed database. It contains entities and associations of interest to the component system. The global conceptual view is the integration of the global external views and is the view of all the data which comprises the manufacturing database. Fragmented views contain occurrences of the global conceptual entities and associations which are partitioned or

replicated across the component systems. Ultimately, any fragmented view is mapped into the particular internal representation of the local subsystems which support it.

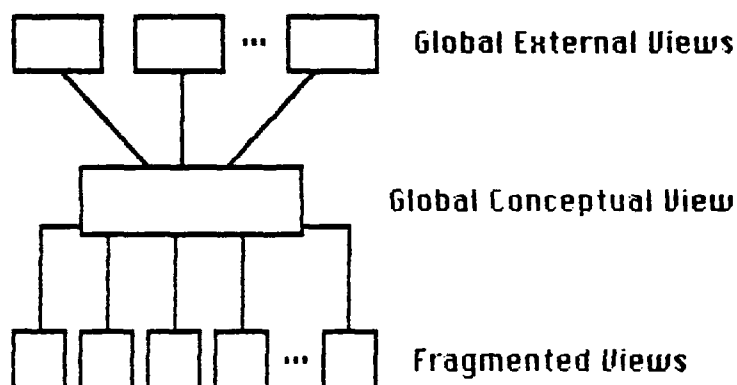


Figure 1: Distributed database architecture.

When a component or user needs to access data in the manufacturing database, it issues a network query, in a common language, against one of the global external views, which then undergoes translation so it can be processed by one or more local data management subsystems. We note here, that a distributed system without a common query language and n local subsystems, requires a total of $(n-1)n$ translators, while an equivalent system with a common query language only requires a total of $2n$ translators.

The work presented in this paper deals with the development of an algorithm required to translate queries from IMDAS' intermediate representation, relational algebra, into QUEL, the query language of the INGRES subsystem.

```

RANGE OF r IS PART
RETRIEVE INTO TEMP (r.ALL) WHERE r.weight < 15

```

Figure 19: Selection example in QUEL.

Union

Given two relations, R and S, which are union compatible, it is possible to derive a relation, T, such that R and S are partitions of T. The union operation can be accomplished in QUEL as shown in Figure 20.

```

RANGE OF r IS R
RANGE OF s IS S
RETRIEVE INTO X (r.ALL)
APPEND TO X (s.ALL)
RANGE OF t IS X
RETRIEVE INTO T (t.ALL)

```

Figure 20: Union in QUEL.

The last two lines in Figure 20 are used to eliminate duplicate tuples which may be present after the execution of the APPEND statement. Duplicate tuples are only allowed in INGRES when relations are stored as heaps, as is the case with temporary relations.

The QUEL sequence for PART1 UNION PART2 is shown in Figure 21.

```

RANGE OF r IS PART1
RANGE OF s IS PART2
RETRIEVE INTO TEMP1 (r.ALL)
APPEND TO TEMP1 (s.ALL)
RANGE OF t IS TEMP1
RETRIEVE INTO TEMP2 (t.ALL)

```

Figure 21: Union example in QUEL.

```
RANGE OF r IS R
RETRIEVE INTO S (r.A1, r.A2, . . . , r.Ak)
```

Figure 16: Projection in QUEL.

The QUEL sequence for the example, PART[color, partno], presented in Chapter II, is shown in Figure 17.

```
RANGE OF r IS PART
RETRIEVE INTO TEMP (r.color, r.partno)
```

Figure 17: Projection example in QUEL.

Selection

Given a relation, R, it is possible to derive a relation, S, such that its tuples are a horizontal subset of R, for which the selection criterion, Q, is true. Q is a logical predicate which compares at least one attribute of R with another attribute of R or with a constant. The selection operation can be accomplished in QUEL with the sequence shown in Figure 18.

```
RANGE OF r IS R
RETRIEVE INTO S (r.ALL) WHERE Q
```

Figure 18: Selection in QUEL.

The QUEL sequence for PART WHERE PART.weight < 15 is shown in Figure 19.

CHAPTER III ALGEBRAIC OPERATIONS WITH QUEL

This chapter presents the QUEL sequences necessary to accomplish all the algebraic operations presented in Chapter II. QUEL, the query language of INGRES, is based on the relational calculus and is relationally complete as proven in [Cod72]. The syntax for QUEL is found in [Sto76] and [Woo81], and is summarized in graphical form in the Appendix, a subset of [RTI84]. Although QUEL is relationally complete, it is not always possible to represent each algebraic operation with a single QUEL statement. The proof that the QUEL sequences presented in this chapter accomplish the relational operations is not presented but can be derived from the material presented in [Ull82].

Projection

Given a relation, R , with n attributes, it is possible to derive a relation, S , such that its tuples are defined on a vertical subset of attributes, A_1, A_2, \dots, A_k , belonging to R . The projection operation can be accomplished in QUEL as shown in Figure 16.

```

(p1, nut, red, 12, London, s1, Smith, 20, London)
(p1, nut, red, 12, London, s4, Clark, 20, London)
(p2, bolt, green, 17, Paris, s2, Jones, 10, Paris)
(p2, bolt, green, 17, Paris, s3, Blake, 30, Paris)
(p4, screw, red, 14, London, s1, Smith, 20, London)
(p4, screw, red, 14, London, s4, Clark, 20, London)
(p5, cam, blue, 12, Paris, s2, Jones, 10, Paris)
(p5, cam, blue, 12, Paris, s3, Blake, 30, Paris)
(p6, cog, red, 19, London, s1, Smith, 20, London)
(p6, cog, red, 19, London, s4, Clark, 20, London)

```

Figure 14: Tuples of PART JOIN SUPPLIER WHERE
PART.city = SUPPLIER.city.

For example, PART NJOIN SUPPLIER OVER city yields the
tuples shown in Figure 15 which is similar to

PART JOIN SUPPLIER WHERE PART.city = SUPPLIER.city
except that the attribute, city, only appears once.

```

(p1, nut, red, 12, London, s1, Smith, 20)
(p1, nut, red, 12, London, s4, Clark, 20)
(p2, bolt, green, 17, Paris, s2, Jones, 10)
(p2, bolt, green, 17, Paris, s3, Blake, 30)
(p4, screw, red, 14, London, s1, Smith, 20)
(p4, screw, red, 14, London, s4, Clark, 20)
(p5, cam, blue, 12, Paris, s2, Jones, 10)
(p5, cam, blue, 12, Paris, s3, Blake, 30)
(p6, cog, red, 19, London, s1, Smith, 20)
(p6, cog, red, 19, London, s4, Clark, 20)

```

Figure 15: Tuples of PART NJOIN SUPPLIER OVER city.

```

(s1, Smith 20, London, p1, nut, red, 12, London)
(s1, Smith 20, London, p2, bolt, green, 17, Paris)
(s1, Smith 20, London, p3, screw, blue, 17, Rome)
(s1, Smith 20, London, p4, screw, red, 14, London)
(s1, Smith 20, London, p5, cam, blue, 12, Paris)
(s1, Smith 20, London, p6, cog, red, 19, London)
(s2, Jones, 10, Paris, p1, nut, red, 12, London)
(s2, Jones, 10, Paris, p2, bolt, green, 17, Paris)
(s2, Jones, 10, Paris, p3, screw, blue, 17, Rome)
(s2, Jones, 10, Paris, p4, screw, red, 14, London)
(s2, Jones, 10, Paris, p5, cam, blue, 12, Paris)
(s2, Jones, 10, Paris, p6, cog, red, 19, London)
(s3, Blake, 30, Paris, p1, nut, red, 12, London)
(s3, Blake, 30, Paris, p2, bolt, green, 17, Paris)
(s3, Blake, 30, Paris, p3, screw, blue, 17, Rome)
(s3, Blake, 30, Paris, p4, screw, red, 14, London)
(s3, Blake, 30, Paris, p5, cam, blue, 12, Paris)
(s3, Blake, 30, Paris, p6, cog, red, 19, London)
(s4, Clark, 20, London, p1, nut, red, 12, London)
(s4, Clark, 20, London, p2, bolt, green, 17, Paris)
(s4, Clark, 20, London, p3, screw, blue, 17, Rome)
(s4, Clark, 20, London, p4, screw, red, 14, London)
(s4, Clark, 20, London, p5, cam, blue, 12, Paris)
(s4, Clark, 20, London, p6, cog, red, 19, London)
(s5, Adams, 30, Athens, p1, nut, red, 12, London)
(s5, Adams, 30, Athens, p2, bolt, green, 17, Paris)
(s5, Adams, 30, Athens, p3, screw, blue, 17, Rome)
(s5, Adams, 30, Athens, p4, screw, red, 14, London)
(s5, Adams, 30, Athens, p5, cam, blue, 12, Paris)
(s5, Adams, 30, Athens, p6, cog, red, 19, London)

```

Figure 13: Tuples of SUPPLIER TIMES PART.

```

(p1, nut, red, 12, London, s1, Smith, 20, London)
(p1, nut, red, 12, London, s2, Jones, 10, Paris)
(p1, nut, red, 12, London, s3, Blake, 30, Paris)
(p1, nut, red, 12, London, s4, Clark, 20, London)
(p1, nut, red, 12, London, s5, Adams, 30, Athens)
(p2, bolt, green, 17, Paris, s1, Smith, 20, London)
(p2, bolt, green, 17, Paris, s2, Jones, 10, Paris)
(p2, bolt, green, 17, Paris, s3, Blake, 30, Paris)
(p2, bolt, green, 17, Paris, s4, Clark, 20, London)
(p2, bolt, green, 17, Paris, s5, Adams, 30, Athens)
(p3, screw, blue, 17, Rome, s1, Smith, 20, London)
(p3, screw, blue, 17, Rome, s2, Jones, 10, Paris)
(p3, screw, blue, 17, Rome, s3, Blake, 30, Paris)
(p3, screw, blue, 17, Rome, s4, Clark, 20, London)
(p3, screw, blue, 17, Rome, s5, Adams, 30, Athens)
(p4, screw, red, 14, London, s1, Smith, 20, London)
(p4, screw, red, 14, London, s2, Jones, 10, Paris)
(p4, screw, red, 14, London, s3, Blake, 30, Paris)
(p4, screw, red, 14, London, s4, Clark, 20, London)
(p4, screw, red, 14, London, s5, Adams, 30, Athens)
(p5, cam, blue, 12, Paris, s1, Smith, 20, London)
(p5, cam, blue, 12, Paris, s2, Jones, 10, Paris)
(p5, cam, blue, 12, Paris, s3, Blake, 30, Paris)
(p5, cam, blue, 12, Paris, s4, Clark, 20, London)
(p5, cam, blue, 12, Paris, s5, Adams, 30, Athens)
(p6, cog, red, 19, London, s1, Smith, 20, London)
(p6, cog, red, 19, London, s2, Jones, 10, Paris)
(p6, cog, red, 19, London, s3, Blake, 30, Paris)
(p6, cog, red, 19, London, s4, Clark, 20, London)
(p6, cog, red, 19, London, s5, Adams, 30, Athens)

```

Figure 12: Tuples of PART TIMES SUPPLIER.

Conversely, SUPPLIER TIMES PART contains the tuples shown in Figure 13.

Note that the product is not commutative since PART TIMES SUPPLIER is not equal to SUPPLIER TIMES PART.

Join

The join of R and S, denoted by R JOIN S, is a subset of the product obtained by selecting tuples from the product which satisfy the join predicate. The join predicate is a qualification statement which contains at least one pair of attributes from each of the operand relations, connected by one of the relational comparison operators: equal, greater than, less than, not equal, less than or equal, and greater than or equal.

For example, PART JOIN SUPPLIER WHERE PART.city = SUPPLIER.city yields the tuples shown in Figure 14. Note that the attribute city appears twice in the result.

Natural Join

The natural join of R and S, denoted by R NJOIN S, is a special case of the join operation. The natural join differs from the join in that only the equality comparison is used in the predicate and duplicate attributes referenced in the predicate are omitted--all the other columns not specifically referenced in the predicate are not omitted.

For example, PART MINUS PART1 yields the tuples shown in Figure 10. Conversely, PART1 MINUS PART yields an empty set, since all the tuples from PART1 also exist in PART.

```
(p1, nut, red, 12, London)
(p4, screw, red, 14, London)
(p5, cam, blue, 12, Paris)
```

Figure 10: Tuples of PART MINUS PART1.

Product

Given R and S, relations of arity k_1 and k_2 respectively, the product of R and S, denoted R TIMES S, is a relation of arity $k_1 + k_2$ containing the set of tuples whose first k_1 components form a tuple in R and whose last k_2 components form a tuple in S.

For example, if the relation SUPPLIER defined as:

```
SUPPLIER(supno, sname, status, city)
```

contains the tuples shown in Figure 11, then PART TIMES SUPPLIER contains the tuples shown in Figure 12.

```
(s1, Smith, 20, London)
(s2, Jones, 10, Paris)
(s3, Blake, 30, Paris)
(s4, Clark, 20, London)
(s5, Adams, 30, Athens)
```

Figure 11: Tuples of SUPPLIER.

Intersection

If R and S are two union compatible relations of arity k, then the intersection of R and S, denoted by $R \text{ INTERSECTION } S$, is the set of k-tuples which belong to both R and S.

For example, if PART3 contains the tuples shown in Figure 8 then $\text{PART2 INTERSECTION PART3}$ yields the tuples shown in Figure 9.

```
(p1, nut, red, 12, London)
(p5, cam, blue, 12, Paris)
(p7, shaft, black, 20, Brussels)
(p8, lug, brown, 15, Paris)
(p10, washer, red, 10, London)
```

Figure 8: Tuples of PART3.

```
(p1, nut, red, 12, London)
(p5, cam, blue, 12, Paris)
```

Figure 9: Tuples of $\text{PART2 INTERSECTION PART3}$.

Difference

If R and S are two union compatible relations of arity k, then the difference between relations R and S, denoted $R \text{ MINUS } S$, is the set of all tuples which are contained in R but not in S. Conversely, $S \text{ MINUS } R$, is the set of all tuples which are contained in S but not in R. It follows that DIFFERENCE is non-commutative, i.e. $S \text{ MINUS } R \neq R \text{ MINUS } S$.

when referencing attributes of the same name from different relations.

Union

The union of relations R and S, denoted R UNION S, is the set of tuples that belong to R, S, or both. The union operation can only be performed when both relations are union compatible, that is, they have equal arity and corresponding attributes are drawn on compatible domains. The result has the same arity as the operands and its attribute names are the same as those found in the first operand.

For example, if PART1 = PART WHERE PART.weight > 15 contains the tuples shown in Figure 6a and PART2 = PART WHERE PART.weight < 15 contains the tuples shown in Figure 6b, then PART1 UNION PART2 has the tuples shown in Figure 7.

(p2, bolt, green, 17, Paris)	(p1, nut, red, 12, London)
(p3, screw, blue, 17, Rome)	(p4, screw, red, 14, London)
(p6, cog, red, 19, London)	(p5, cam, blue, 12, Paris)
(a)	(b)

Figure 6: Tuples of PART1 and PART2.

```
(p2, bolt, green, 17, Paris)
(p3, screw, blue, 17, Rome)
(p6, cog, red, 19, London)
(p1, nut, red, 12, London)
(p4, screw, red, 14, London)
(p5, cam, blue, 12, Paris)
```

Figure 7: Tuples of PART1 UNION PART2.

For example, `PART[color, partno]` yields the tuples shown in Figure 4.

```
(red, p1)
(green, p2)
(blue, p3)
(red, p4)
(blue, p5)
(red, p6)
```

Figure 4: Tuples of `PART[color, partno]`.

Selection

Given a relation `R` of arity `k`, a horizontal subset can be defined on `R`, such that the occurrences of certain components of the subset satisfy a logical predicate which references only attributes of `R` [Ull82]. The selection on `R` is denoted `R WHERE p`, in which `p` is the logical predicate and is also known as the selection criterion.

For example, using the `PART` relation, `PART WHERE PART.weight < 15` corresponds to the tuples of parts which weigh less than 15 units, i.e. the tuples shown in Figure 5.

```
(p1, nut, red, 12, London)
(p4, screw, red, 14, London)
(p5, cam, blue, 12, Paris)
```

Figure 5: Tuples of `PART WHERE PART.weight < 15`.

Note that the attributes of a relation are referenced using qualified names, i.e. the relation name, followed by a period, followed by the actual attribute name. The purpose of using qualified names is to minimize ambiguity, such as,

arity 5, could consist of the tuples shown in Figure 3 and is expressed empirically as:

PART (partno, pname, color, weight, city)

where PART is defined on the f-mappings of Figure 2.

```
(p1, nut, red, 12, London)
(p2, bolt, green, 17, Paris)
(p3, screw, blue, 17, Rome)
(p4, screw, red, 14, London)
(p5, cam, blue, 12, Paris)
(p6, cog, red, 19, London)
```

Figure 3: Tuples of relation PART.

The algebraic operators which are used to manipulate relations are: selection, projection, union, intersection, difference, product, join, and natural join. Depending on the number of operands, these algebraic operators are classified into unary and binary. Projection and selection are unary, i.e. they use one operand. The other operators are binary, i.e. they use two operands.

Projection

Given a relation R of arity k , the projection of one or more components of R , denoted by $R[i_1, i_2, \dots, i_m]$, where $m \leq k$, is the set of m -tuples, $a_1 a_2 \dots a_m$ such that there is some k -tuple $b_1 b_2 \dots b_k$ in R for which $a_j = b_{i_j}$ for $j = 1, 2, \dots, m$ [Ull82]. In simpler terms, we take a relation, remove some of its columns and/or permute some of the remaining columns.

CHAPTER II RELATIONAL ALGEBRA

The purpose of this chapter is to present the relational algebra notation that will be used in future discussions leading to the translation algorithm. The presentation of this notation is meant to be brief and does not cover all aspects of the relational algebra.

A tuple is defined in [Ull82] as a mapping from attribute names to values in the domains of the attributes. An example of a tuple is:

(p1, nut, red, 12, London)

and the mapping f which defines that tuple is shown in Figure 2.

```
f(partno) = p1
f(pname)  = nut
f(color)  = red
f(weight) = 12
f(city)   = London
```

Figure 2: The mapping f .

Tuples can be grouped into relations. A relation is defined in [Ull82] as a set of k -tuples, where k is fixed and is known as the arity of the relation. The arity of the tuple in the above example is 5. A relation, PART, of

Network queries made in a global data manipulation language (GDML) are transformed into relational algebra query trees which are in turn partitioned for processing by the pertinent subsystems. The translation algorithm, to be described in this paper, maps the relational algebra into QUEL. The significance of this work is that it lays a foundation to develop other translation algorithms which can be used to map the GDML into other subsystems. Relevant work dealing with this type of translation is found in [Cer in press]. Material which justifies the use of relational algebra as IMDAS' intermediate representation can be found in [Klu80] and [Su81].

The work is organized as follows. The material in Chapter II, based on [Cod72], [Dat82], and [Ull82], introduces the notation and definitions of the relational algebra. Chapter III develops the QUEL statements which accomplish all the algebraic operators using [Ull82] and [Woo81]. Chapter IV develops a node-by-node algorithm to translate relational algebra query trees into QUEL. Using [Chu82], Chapter V discusses minimization and presents an optimized translation algorithm. Chapter VI discusses implementation of the latter algorithm. Chapter VII gives the concluding remarks about this work.

Intersection

Given two relations, R and S, which are union compatible, it is possible to derive a relation, T, such that all its tuples exist, both, in R and in S. The intersection operation can be accomplished in QUEL as shown in Figure 22. Note that A1, A2, . . . Ai are the attributes of R, and B1, B2, . . . Bi are the attributes of S.

```
RANGE OF r IS R
RANGE OF s IS S
RETRIEVE INTO T (r.ALL) WHERE r.A1 = s.B1
AND r.A2 = s.B2 . . . AND r.Ai = s.Bi
```

Figure 22: Intersection in QUEL.

The QUEL sequence for PART2 INTERSECTION PART3 is shown in Figure 23.

```
RANGE OF r IS PART1
RANGE OF s IS PART3
RETRIEVE INTO TEMP (r.ALL)
WHERE r.partno = s.partno
AND r.pname = s.pname
AND r.color = s.color
AND r.weight = s.weight
AND r.city = s.city
```

Figure 23: Intersection example in QUEL.

Note that comparisons using all corresponding attribute name pairs are built into the WHERE clause to test for tuple equivalency.

Difference

Given two relations, R and S, which are union compatible, it is possible to derive a difference relation, T, such that all its tuples, except those tuples which already exist in S, are taken from R. The difference operation can be accomplished in QUEL as shown in Figure 24. Note that A1, A2, . . . Ai are the attributes of R, and B1, B2, . . . Bi are the attributes of S.

```
RANGE OF r IS R
RANGE OF s IS S
RETRIEVE INTO T (r.ALL)
RANGE OF t IS T
DELETE t WHERE t.A1 = s.B1
AND t.A2 = s.B2 . . . AND t.Ai = s.Bi
```

Figure 24: Difference in QUEL.

The QUEL sequence for PART MINUS PART1 is shown in Figure 25.

```
RANGE OF r IS PART
RANGE OF s IS PART1
RETRIEVE INTO TEMP (r.ALL)
RANGE OF t IS TEMP
DELETE t WHERE t.partno = s.partno
AND t.pname = s.pname
AND t.color = s.color
AND t.weight = s.weight
AND t.city = s.city
```

Figure 25: Difference example in QUEL.

Note that tuple equivalency is achieved the same way as in the intersection operation.

Product

Given two relations, R and S, it is possible to derive a relation, T, such that all its tuples are formed by making all possible combinations taking one tuple from R and one tuple from S and concatenating them. The product operation can be accomplished in QUEL as shown in Figure 26.

```
RANGE OF r IS R
RANGE OF s IS S
RETRIEVE INTO T (r.ALL, s.ALL)
```

Figure 26: Product in QUEL.

The QUEL sequence for PART TIMES SUPPLIER is shown in Figure 27.

```
RANGE OF r IS PART
RANGE OF s IS SUPPLIER
RETRIEVE INTO TEMP (r.ALL, s.supno,
s.sname, s.status, city2 = s.city)
```

Figure 27: Product example in QUEL.

Note that when duplicate attribute names are present, e.g. city, they must be aliased to a unique name in the result. This is because INGRES does not allow duplicate attribute names within the same relation. If all the attribute names of both relations had been different, the target list in the RETRIEVE statement would have been: (r.ALL, s.ALL).

Conversely, the QUEL sequence for SUPPLIER TIMES PART is shown in Figure 28.

```

RANGE OF r IS SUPPLIER
RANGE OF s IS PART
RETRIEVE INTO TEMP (r.ALL, s.partno,
s.pname, s.color, s.weight, city2 = s.city)

```

Figure 28: Product example in QUEL.

The RETRIEVE statement could be stated equivalently as:

```

RETRIEVE INTO TEMP (r.supno, r.sname, r.status, city2 =
r.city, s.ALL).

```

Join

Given two relations, R and S, it is possible to derive a relation, T, such that all its tuples belong to a subset of the cartesian product which satisfy a join condition, P. P is a logical predicate which references at least one pair of attributes, one from each relation, and compares them using any of the relational operators as described in Chapter II. The join operation can be accomplished in QUEL as shown in Figure 29.

```

RANGE OF r IS R
RANGE OF s IS S
RETRIEVE INTO T (r.ALL, s.ALL) WHERE P

```

Figure 29: Join in QUEL.

The QUEL sequence for PART JOIN SUPPLIER WHERE PART.city = SUPPLIER.city is shown in Figure 30.


```

RANGE OF r IS PART
RANGE OF s IS SUPPLIER
RETRIEVE INTO TEMP (r.ALL, s.supno,
s.sname, s.status, city2 = s.city)
WHERE r.city = s.city

```

Figure 30: Join example in QUEL.

As in the product, duplicate attribute names appear aliased in the result.

Natural Join

Given two relations, R and S, it is possible to derive a relation, T, such that all its tuples belong to the subset of the cartesian product which satisfy a join condition P. P is a logical predicate which references at least one pair of attributes, one from each relation, and compares them using only the equality operator. Attribute names referenced in the join condition only appear once in the result. The natural join operation can be accomplished in QUEL as shown in Figure 31.

```

RANGE OF r IS R
RANGE OF s IS S
RETRIEVE INTO TEMP (r.ALL, s.ALL)
WHERE P

```

Figure 31: Natural Join in QUEL.

The QUEL sequence for PART NJOIN SUPPLIER OVER city is shown in Figure 32.

```
RANGE OF r IS PART  
RANGE OF s IS SUPPLIER  
RETRIEVE INTO TEMP (r.ALL, s.supno,  
s.sname, s.status)  
WHERE r.city = s.city
```

Figure 32: Natural join example in QUEL.

translation can be performed without accessing the target subsystem.

The algorithm has two main components: tree processing and operation mapping. Tree processing determines the correct order of execution for the operations contained in the query tree. Operation mapping converts algebraic operations specified in the query tree into a sequence of QUEL statements. It is based on the QUEL sequences that were presented in Chapter III.

Tree Processing

Each operation specified in a query tree node will use as operands relations contained in the target subsystem and/or results of operations from preceding nodes. The resulting QUEL sequence provides a way to store intermediate results from lower level nodes so they can be reused at higher levels. This is accomplished through the use of temporary relations. For the purposes of this discussion, temporary relations will be referred to as TEMPx relations, x being a sequence number which is generated by the translator. These TEMPx relations are destroyed at some point in the execution of the QUEL sequence, when they are no longer needed, e.g. prior to translation of another query packet.

The correct order of execution is determined by examining the sibling pointers at each node. This order is noted, e.g. using a stack, and then used to generate correctly ordered QUEL statements. Figure 33a shows an

CHAPTER IV TRANSLATION ALGORITHM

The algorithm which describes the steps necessary to convert relational algebra query trees into QUEL statements is discussed in this chapter. The algorithm presented, can be used to implement a translator program which generates QUEL statements that are executable on the INGRES subsystem. The translator takes as input a query packet file in a predefined format and generates the proper sequence of QUEL commands to be executed by the target subsystem. The query packet specifies a series of algebraic operations organized as a binary tree structure in which the order of precedence is from bottom to top.

The way in which the query tree is stored is arbitrary and depends on implementation design, thus it is not discussed here. It is important to note, however, that it contains pointers which allow the translator to discern the relationships between all its nodes. It must also contain information which defines the algebraic operations to be performed at each node, i.e. operation, operands, attributes, conditions, etc. It must also include dictionary information about all operands, so that the entire

example of a query tree. Figures 33b and 33c show possible orders of execution. The difference between the stacks is the side chosen first during traversal. The left branch is chosen first in Figure 33b, while the right branch is chosen first in Figure 33c.

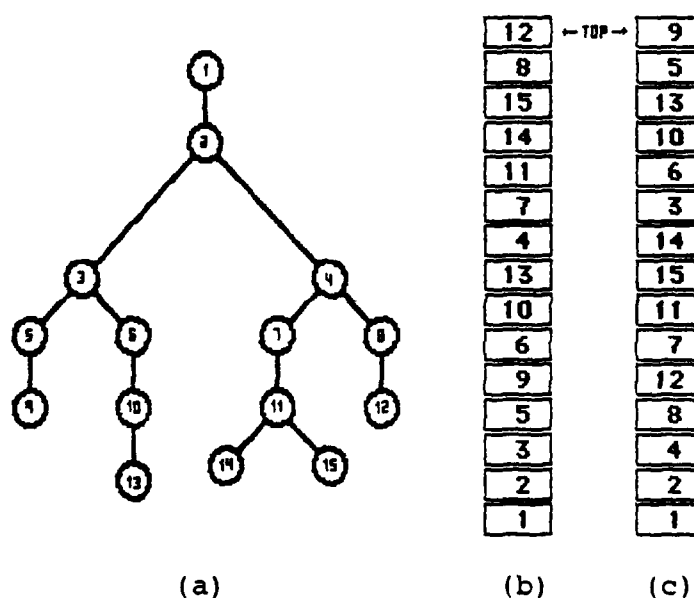


Figure 33: Query tree and execution stacks.

Operation Mapping

This portion of the algorithm translates each node operation into the corresponding sequence of QUEL statements. In order to generate the appropriate sequence for a given node, the algorithm uses predefined templates for each algebraic operation, i.e. each algebraic operator

has a fixed template associated with it. These templates closely resemble the sequences presented in Chapter III.

To translate a node operation into the corresponding QUEL sequence, the template is copied into a work buffer and the details which are particular to that node, i.e. sequence numbers, relation names, attribute names, etc., are inserted into the appropriate fields of the buffer, thus building up the appropriate sequence for that node. Once the sequence has been built, the contents of the buffer can be saved, e.g. appended to an output file.

Having a template for each operation defines a unique series of programmed steps required to translate the query tree to QUEL on a node-per-node basis. This method is relatively simple to implement although it may not be optimal. The node-per-node translation algorithm is presented to acquaint the reader with the main details of the translation process. An optimal method of translation is presented in the next chapter.

In the coming paragraphs, the following algebraic operators are mapped into QUEL: Union, Intersection, Difference, Selection, Projection, Product, Join, and Natural Join.

Projection

The steps necessary to translate a node containing a projection operation are shown in Figure 34.

Load Projection template into buffer area;

```
RANGE OF L IS relname
RETRIEVE INTO TEMPx (L.attrlist)
```

Generate sequence number and insert into TEMPx field;

Insert relation name into relname field;

Insert attribute names into target list, using
L-prefixes on all the attribute names of the relation;

Figure 34: Pseudo-code for Projection node.

Selection

The steps necessary to translate a node containing a selection operation are shown in Figure 35.

Load Selection template into buffer area;

```
RANGE OF L IS relname
RETRIEVE INTO TEMPx (L.ALL) WHERE P
```

Generate sequence number and insert into TEMPx field;

Insert relation name into relname field;

Insert selection predicate into P field;

Figure 35: Pseudo-code for Selection node.

Union

The steps necessary to translate a node containing a projection operation are presented in Figure 36.

Load Union template into buffer area;

```
RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.ALL)
APPEND TO TEMPx (R.ALL)
RANGE OF X IS TEMPx
RETRIEVE INTO TEMPy (X.ALL)
```

Generate sequence number and insert into TEMPx and TEMPy fields;

Insert relation names into relname1 and relname2 fields;

Figure 36: Pseudo-code for Union node.

Intersection

The steps necessary to translate a node containing a union operation are shown in Figure 37.

Load Intersection template into buffer area;

```
RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.ALL)
WHERE L.attr1 = R.attr1 AND L.attr2 = R.attr2
AND . . . L.attrN = R.attrN
```

Generate sequence number and insert into TEMPx field;

Insert relation names into relname1 and relname2 fields;

Build WHERE clause using L-prefixes with the attribute names of the left relation and R-prefixes with the attribute names of the right relation;

Figure 37: Pseudo-code for Intersection node.

Difference

The steps necessary to translate a node containing a difference operation are shown in Figure 38.

Load Difference template into buffer area;

```
RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO Tempx (L.ALL)
RANGE OF L IS TEMPx
DELETE L WHERE L.attr1 = R.attr1 AND L.attr2 =
R.attr2 . . . AND L.attrN = R.attrN
```

Generate sequence number and insert into TEMPx fields;

Insert relation names into fields relname1 and relname2;

Build WHERE clause using L-prefixes with the attribute names of the left relation and R-prefixes with the attribute names of the right relation;

Figure 38: Pseudo-code for Difference node.

Product

The steps necessary to translate a node containing a product operation are shown in Figure 39.

Load Product template into buffer area;

```
RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.attrlist1, R.attrlist2)
```

Generate sequence number and insert into TEMPx field;

Insert relation names into relname1 and relname2 fields;

Build target list using L-prefixes with all the attribute names of the left relation and R-prefixes with all the attribute names of the right relation;

For duplicate attribute names, generate alias names and enter them into the aliases table;

Figure 39: Pseudo-code for Product node.

Join

The steps necessary to translate a node containing a join operation are shown in Figure 40.

Load Join template into buffer area;

```
RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.attrlist1, R.attrlist2)
WHERE PJ
```

Generate sequence number and insert into TEMPx field;

Insert relation names into relname1 and relname2 fields;

Build target list using L-prefixes with all the attribute names of the left relation and R-prefixes with all the attribute names of the right relation;

For duplicate attribute names generate alias names and enter them into the aliases table;

Build WHERE clause using the join predicate and add L-prefixes to the attribute names of the left relation and R-prefixes to the attribute names of the right relation;

Figure 40. Pseudo-code for Join node.

Natural Join

The steps necessary to translate a node containing a natural join operation are shown in Figure 41.

Join

The steps necessary to translate a Type II node containing the join operation are shown in Figure 53.

Load Difference template into buffer area;

```

RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.attrlistL) WHERE PL

RANGE OF T IS TEMPx
DELETE T WHERE T.attrL1 = R.attrR1 AND T.attrL2 =
R.attrR2 . . . AND L.attrLn = R.attrRn AND PR

```

Generate sequence number and insert it into the TEMPx fields;

Insert relation names into relname1 and relname 2 fields;

Build target list for RETRIEVE statement using L-prefixes with all the attribute names of the left topmost attribute list;

Build first WHERE clause using using L-prefixes with all the attribute names in the left composite predicate;

Build second WHERE clause using T-prefixes on all the attribute names of the left topmost attribute list and R-prefixes on all the attribute names of the right topmost attribute list;

Add R-prefixes to the attribute names of the right composite predicate and append to the second WHERE clause;

Figure 51: Pseudo-code for Type II node with Difference.

Load Intersection template into buffer area;

```

RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.attrlistL)

WHERE L.attrL1 = R.attrR1 AND L.attrL2 =
R.attrR2 . . . AND L.attrLn = R.attrRn

AND PL AND PR

```

Generate sequence number and insert it into the TEMPx field;

Insert relation names into the relname1 and relname2 fields;

Build target list for RETRIEVE statement using L-prefixes with all the attribute names of the left topmost attribute list;

Select corresponding attribute names from the right topmost and left topmost attribute lists and append to WHERE clause using the proper L- and R-prefixes;

Build WHERE clause using L-prefixes with the attribute names in the left composite predicate and R-prefixes on the attribute names in the right composite predicate;

Figure 50: Psuedo-code for Type II node with Intersection.

Difference

The steps necessary to translate a Type II node containing the difference operation are shown in Figure 51.

Product

The steps necessary to translate a Type II node containing the product operation are shown in Figure 52.

Load Union template into buffer area;

```

RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.attrlistL) WHERE PL

APPEND TO TEMPx (attrL1 = L.attrR1, attrL2 =
L.attrR2, . . . attrLn = L.attrRn)

WHERE PR

RANGE OF T IS TEMPx
RETRIEVE INTO TEMPy (T.ALL)

```

Generate sequence numbers and insert them into the TEMPx and TEMPy fields;

Insert relation names into the relname1 and relname2 fields;

Build target list for RETRIEVE statement using L-prefixes on all the attribute names in the left topmost attribute list;

Build first WHERE clause using L-prefixes on the attribute names in left composite predicate;

Build target list for APPEND statement using R-prefixes on all the attribute names of the right topmost attribute list aliased to all the attribute names of the left topmost attribute list;

Build first WHERE clause using R-prefixes on the attribute names in the right composite predicate;

Figure 49: Pseudo-code for Type II node with Union.

Intersection

The steps necessary to translate a Type II node containing the intersection operation are shown in Figure 50.

RANGE OF L IS RELATION1

RANGE OF R IS RELATION2

RETRIEVE INTO RESULT (L.left-topmost-attrlist,
R.right-topmost-attrlist)

WHERE join-predicate AND left-composite-predicate
AND right-composite-predicate

Natural Join. When two Type 1 branches are combined through the natural join operation, it is equivalent to the following:

RANGE OF L IS RELATION1

RANGE OF R IS RELATION2

RETRIEVE INTO RESULT (L.left-topmost-attrlist,
R.right-topmost-attrlist)

WHERE join-predicate AND left-composite-predicate
AND right-composite-predicate

Optimal Algorithm

We shall now present the steps for query translation for each of the binary operations which use the minimization rules just shown.

Union

The steps necessary to translate a Type II node containing the union operation are shown in Figure 49.

AND L.attr1 = R.attr1 AND L.attr2 = R.attr2

AND . . . L.attrn = R.attrn

Difference. When two Type 1 branches are combined through the difference operation, it is equivalent to the following:

RANGE OF L IS RELATION1

RANGE OF R IS RELATION2

RETRIEVE INTO RESULT (L.left-topmost-attrlist)

WHERE left-composite-predicate

RANGE OF T IS RESULT

DELETE T WHERE right-composite-predicate AND

L.attr1 = R.attr1 AND L.attr2 = R.attr2

AND . . . AND L.attrn = R.attrn

Product. When two Type 1 branches are combined through the product operation, it is equivalent to the following:

RANGE OF L IS RELATION1

RANGE OF R IS RELATION2

RETRIEVE INTO RESULT (L.left-topmost-attrlist,

R.right-topmost-attrlist)

WHERE left-composite-predicate AND

right-composite-predicate

Join. When two Type 1 branches are combined through the join operation, it is equivalent to the following:

RANGE OF X IS R
 RETRIEVE INTO TEMP (X.topmost-attrlist)
 WHERE composite-predicate

Type II

This type of pattern has one variation for each of the binary operations.

Union. When two Type 1 branches are combined through the union operation, it is equivalent to the following:

RANGE OF L IS RELATION1
 RANGE OF R IS RELATION2
 RETRIEVE INTO TEMP (L.left-topmost-attrlist)
 WHERE left-composite-predicate
 APPEND TO TEMP (R.right-topmost-attrlist)
 WHERE right-composite-predicate
 RANGE OF T IS TEMP
 RETRIEVE INTO RESULT (T.ALL)

Intersection. When two Type 1 branches are combined through the intersection operation, it is equivalent to the following:

RANGE OF L IS RELATION1
 RANGE OF R IS RELATION2
 RETRIEVE INTO RESULT (L.left-topmost-attrlist)
 WHERE left-composite-predicate AND
 right-composite-predicate

RETRIEVE INTO TEMP_n (X.attrlist n)

is equivalent to

RANGE OF X IS R

RETRIEVE INTO TEMP (X.attrlist n)

Consecutive selections. Any number of consecutive selections on a relation R is equivalent to one selection which uses a selection predicate which is the conjunction of all their predicates. This predicate is referred to as the composite predicate. Thus,

RANGE OF X IS R

RETRIEVE INTO TEMP₁ (X.ALL) WHERE P₁

RANGE OF X IS TEMP₁

RETRIEVE INTO TEMP₂ (X.ALL) WHERE P₂

:

RANGE OF X IS TEMP_{m-1}

RETRIEVE INTO TEMP_m (X.ALL) WHERE P_m

is equivalent to

RANGE OF X IS R

RETRIEVE INTO TEMP WHERE P₁ AND P₂ AND . . . AND P_m

when P₁, P₂, . . . , P_m are predicates which reference only attributes of R.

Mixed projections and selections. Any mix of selections and projections on relation R, is equivalent to one projection which uses the topmost attribute list and one selection which uses the composite predicate, as follows:

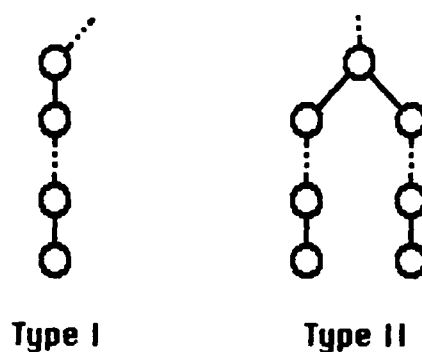


Figure 48: Basic pattern types.

Type 1

The two unary operations, projection and selection, give rise to three variations of this pattern: consecutive projections only, consecutive selections only, and mixed selections and projections.

Consecutive projections. Any number of consecutive projections on a relation R , such that the set of attributes specified in any projection is always a subset of the set of attributes specified in the previous projection, is equivalent to one projection on R using the attribute set of the last projection. This attribute set is referred to as the topmost attribute list. Thus,

```

RANGE OF X IS R
RETRIEVE INTO TEMP1 (X.attrlist 1)
RANGE OF X IS TEMP1
RETRIEVE INTO TEMP2 (X.attrlist 2)
:
RANGE OF X IS TEMPn-1

```

CHAPTER V MINIMIZATION

The node-per-node translation algorithm presented in the previous section can be implemented with relative ease but the large number of QUEL statements that it produces makes it inefficient. By introducing minimization techniques [Chu82], it is possible to produce a smaller number of QUEL statements which accomplish the same results as the node-per-node translation.

Minimization Rules

Compression of the query tree during the tree processing phase yields a new query tree which consists of fewer nodes. The nodes of the new query tree specify combinations of algebraic operations that can be expressed with less QUEL statements. Selection of nodes to be compressed is accomplished by recognition of two basic types of reoccurring patterns, shown in Figure 48.

The first pattern occurs when a series of consecutive unary operations is specified and will be referred to as Type I. The second pattern occurs when the results of two separate Type I patterns are combined through a binary operation and will be referred to as Type II.

```

7  RANGE OF L IS SPJ
   RETRIEVE INTO TEMP1 (L.ALL) WHERE L.supno = "s1"
6  RANGE OF L IS TEMP1
   RETRIEVE INTO TEMP2 (R.partno)
5  RANGE OF L IS PART
   RETRIEVE INTO TEMP3 (L.partno)
4  RANGE OF L IS TEMP3
   RANGE OF R IS TEMP2
   RETRIEVE INTO TEMP4 (L.ALL)
   RANGE OF T IS TEMP4
   DELETE T WHERE T.partno = R.partno
3  RANGE OF L IS SPJ
   RANGE OF R IS TEMP4
   RETRIEVE INTO TEMP5 (L.ALL) WHERE L.partno = R.partno
2  RANGE OF L IS TEMP5
   RETRIEVE INTO TEMP6 (L.projno)
   RANGE OF L IS PROJECT
   RETRIEVE INTO TEMP7 (L.projno)
0  RANGE OF L IS TEMP7
   RANGE OF R IS TEMP6
   RETRIEVE INTO TEMP7 (L.ALL)
   RANGE OF T IS TEMP7
   DELETE T WHERE T.projno = R.projno

```

Figure 47: QUEL translation of Query 3.

```

4  RANGE OF L IS PROJECT
   RETRIEVE INTO TEMP1 (L.projno, L.city)
5  RANGE OF L IS SUPPLIER
   RETRIEVE INTO TEMP2 (L.supno, L.city)
3  RANGE OF L IS TEMP2
   RANGE OF R IS SPJ
   RETRIEVE INTO TEMP3 (L.ALL, supno2 = R.supno, R.partno,
   R.projno, R.qty)
2  RANGE OF L IS TEMP3
   RANGE OF R IS TEMP1
   RETRIEVE INTO TEMP4 (L.ALL, projno2 = R.projno,
   city2 = R.city)
1  RANGE OF L IS TEMP4
   RETRIEVE INTO TEMP5 (L.ALL)
   WHERE L.supno = L.supno2 AND L.projno = L.projno2
0  RANGE OF L IS TEMP5
   RETRIEVE INTO TEMP6 (L.city, L.partno, L.city2)

```

Figure 45: QUEL translation for Query 2.

from supplier S1." The equivalent algebraic representation for this query is:

```

PROJECT[projno] MINUS ((SPJ NJOIN(PART[partno] MINUS
    (SPJ WHERE supno = 'S1') [partno])) [projno])

```

and the query tree and order of traversal are shown in Figure 46. The corresponding QUEL translation is shown in Figure 47.

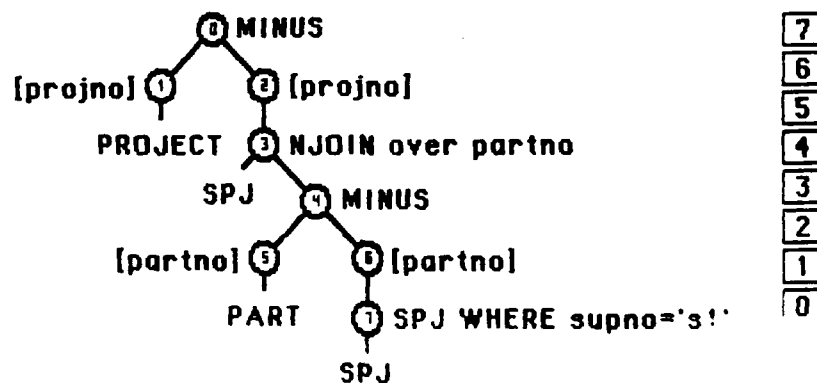


Figure 46: Query tree for Query 3.

Query 2

Exercise 7.19 in [Dat82] read as follows: "Get all (city, partno, city) triples such that a supplier in the first city supplies the specified part to a project in the second city." The equivalent algebraic representation for this query is:

```
((SUPPLIER[supno, city] TIMES SPJ TIMES PROJECT[projno,
city]) WHERE SUPPLIER.supno = SPJ.supno AND
SPJ.projno = PROJECT.projno) [SUPPLIER.city, partno,
PROJECT.city]
```

and the query tree and order of traversal are shown in Figure 44. The corresponding QUEL translation is shown in Figure 45.

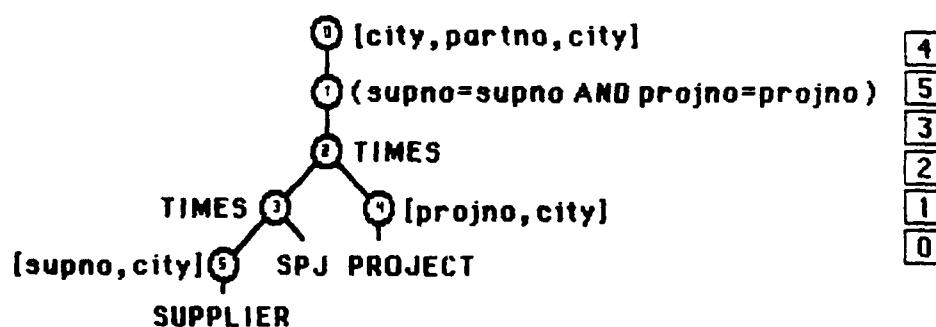


Figure 44: Query tree for Query 2.

Query 3

Exercise 7.25 in [Dat82] read as follows: "Get projno values for projects which use only parts which are available

Query 1

Exercise 7.17 in [Dat82] reads as follows: "Get projno values for projects using at least one part available from supplier S1." The algebraic representation for this query is:

$((SPJ \text{ WHERE } supno = 'S1')[partno] \text{ NJOIN } SPJ)[projno]$

and the query tree and order of traversal are shown in Figure 42. The corresponding QUEL translation is shown in Figure 43.

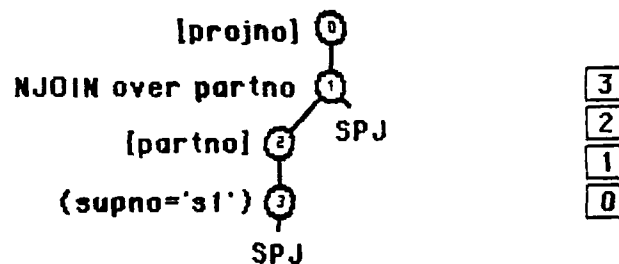


Figure 42: Query tree for Query 1.

```

3  RANGE OF L IS SPJ
   RETRIEVE INTO TEMP1 (L.All) WHERE L.supno = "s1"
2  RANGE OF L IS TEMP1
   RETRIEVE INTO TEMP2 (L.partno)
   RANGE OF L IS TEMP2
1  RANGE OF R IS SPJ
   RETRIEVE INTO TEMP3 (L.partno, R.supno, R.projno, R.qty)
   WHERE L.partno = R.partno
0  RANGE OF L IS TEMP3
   RETRIEVE INTO TEMP4 (L.projno)

```

Figure 43: QUEL translation for Query 1.

Load Natural Join template into buffer area;

```
RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.attrlist1, R.attrlist2)
WHERE PJ
```

Generate sequence number and insert into TEMPx field;

Insert relation names into relname1 and relname2 fields;

Build target list using L-prefixes with all the attribute names of the left relation and R-prefixes with all the attribute names of the right relation;

Delete duplicate attribute names which appear on the right side(s) of the join predicate;

For other duplicate attribute names, generate alias names and enter them into the aliases table;

Build WHERE clause using the join predicate and add L-prefixes to the attribute names of the left relation and R-prefixes to the attribute names of the right relation;

Figure 41: Pseudo-code for Natural Join node.

Examples

The following examples illustrate the mechanics of the translation algorithm and are based on the Parts & Suppliers Database found in [Dat82], shown below:

SUPPLIER (supno, sname, status, city)

PART (partno, pname, color, weight, city)

PROJECT (projno, jname, city)

SPJ (supno, partno, projno, qty)

Load product template into buffer area;

```

RANGE OF L IS relname1
RANGE OF R IS relname 2
RETRIEVE INTO TEMPx (L.attrlistL, R.attrlistR)
WHERE PL AND PR

```

Generate sequence number and insert it into the TEMPx field;

Insert relation names into the relname1 and relname2 fields;

Build target list for RETRIEVE statement using L-prefixes with all the attribute names of the left topmost attribute list and R-prefixes with all the attribute names of the right topmost attribute list;

Alias duplicate attribute names of the right topmost attribute list to local names annotated in aliases table;

Build WHERE clause using L-prefixes with the attribute names in the left composite predicate and R-prefixes with the attribute names in the right composite predicate;

Figure 52: Pseudo-code for Type II node with Product.

Load Join template into buffer area;

```

RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.attrlistL, R.attlistR)
WHERE PJ AND PL AND PR

```

Generate sequence number and insert it into the TEMPx field;

Insert relation names into relname1 and relname2 fields;

Build target list for RETRIEVE statement using L-prefixes with all the attribute names of the left topmost attribute list and R-prefixes with the attribute names of the right topmost attribute list;

Alias duplicate attribute names of the right topmost attribute list to local names annotated in aliases table;

Build WHERE clause using using L-prefixes with attribute names in the left composite predicate, R-prefixes with attribute names in the right composite predicate, and the proper prefixes with the attribute names in the join predicate;

Figure 53: Pseudo-code of Type II node with Join.

Natural Join

The steps necessary to translate a Type II node containing the natural join operation are shown in Figure 54.

Load Natural Join template into buffer area;

```
RANGE OF L IS relname1
RANGE OF R IS relname2
RETRIEVE INTO TEMPx (L.attrlistL, R.attlistR)
WHERE PJ AND PL AND PR
```

Generate sequence number and insert it into the TEMPx field;

Insert relation names into relname1 and relname2 fields;

Build target list for RETRIEVE statement using L-prefixes with all the attribute names of the left topmost attribute list and R-prefixes with the attribute names of the right topmost attribute list;

Delete redundant attribute names, i.e. the attribute names appearing on the right side of the join predicate;

Alias duplicate attribute names of the right topmost attribute list to local names annotated in aliases table;

Build WHERE clause using L-prefixes with attribute names in the left composite predicate, R-prefixes with attribute names in the right composite predicate, and the proper prefixes with the attribute names in the join predicate;

Figure 54: Pseudo-code for Type II node with Natural Join.

Examples

The examples which are presented below illustrate the improvement that results from applying the new translation algorithm to the sample queries presented in the previous chapter. A reduced number of QUEL statements is achieved when using this method of translation.

Query 1

The translation of this query can be optimized by compressing the query tree as shown in Figure 55. The QUEL translation for the compressed tree is shown in Figure 56.

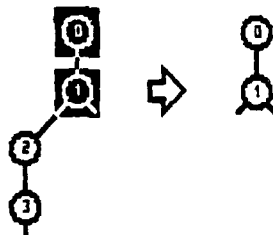


Figure 55: Tree compression for Query 1.

```

1  RANGE OF L IS SPJ
   RANGE OF R IS SPJ
   RETRIEVE INTO TEMP1 (L.partno, R.supno, R.projno, R.qty)
   WHERE L.supno = "s1" AND L.partno = R.partno
0  RANGE OF L IS TEMP1
   RETRIEVE INTO TEMP2 (L.projno)

```

Figure 56: Optimized QUEL translation for Query 1.

Query 2

The translation of this query can be optimized by compressing the query tree as shown in Figure 57. The QUEL translation for the compressed tree is shown in Figure 58.

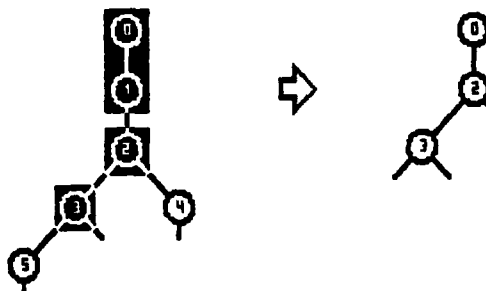


Figure 57: Tree compression for Query 2.

```

3  RANGE OF L IS SUPPLIER
   RANGE OF R IS SPJ
   RETRIEVE INTO TEMP1 (L.supno, L.city,
supno2 = R.supno, R.partno, R.projno, R.qty)
2  RANGE OF L IS TEMP1
   RANGE OF R IS PROJECT
   RETRIEVE INTO TEMP2 (L.ALL, projno2 = R.projno
city2 = R.city)
0  RANGE OF L IS TEMP2
   RETRIEVE INTO TEMP3 (L.city, L.partno, L.city2)
   WHERE L.supno = L.supno2 AND L.projno = L.projno2

```

Figure 58: Optimized QUEL translation for Query 2.

Query 3

The translation of this query can be optimized by compressing the query tree as shown in Figure 59. The QUEL translation for the compressed tree is shown in Figure 60.

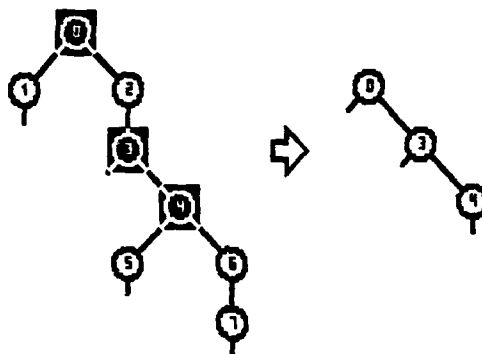


Figure 59: Tree compression for Query 3.

```

4  RANGE OF L IS PART
   RANGE OF P IS SPJ
   RETRIEVE INTO TEMP1 (L.partno)
   RANGE OF T IS TEMP1
   DELETE T WHERE R.supno = "s1" AND T.partno = R.partno
3  RANGE OF L IS SPJ
   RANGE OF R IS TEMP1
   RETRIEVE INTO TEMP2 (L.projno) WHERE L.partno = R.partno
0  RANGE OF L IS PROJECT
   RANGE OF R IS TEMP2
   RETRIEVE INTO TEMP3 (L.projno)
   RANGE OF T IS TEMP3
   DELETE T WHERE T.projno = R.projno

```

Figure 60: Optimized QUEL translation for Query 3.

CHAPTER VI IMPLEMENTATION

This chapter discusses some general guidelines to follow in implementing the algorithm presented in the previous chapter. An implementation has been successfully completed in the "C" language by Mr. Mohamed Khatib, a graduate student currently doing related research for the NBS project at the time of this writing. The main topics that will be discussed are: query packet format, tree processing, operation mapping, and minimization.

Query Packet Format

The Master Data Administrator System (MDAS) of [NBS85] issues data management commands, in the form of query packets, to each of the distributed data management subsystems. The packets are organized into records of predefined format. These records contain all the algebraic operators, relation names, attribute names, and qualifications which completely specify what the subsystem has to do.

Each query packet has three types of records: tree descriptors, operation descriptors, and relation tables. Tree descriptors define the structure of a query tree,

that is, they describe the tree in terms of node inter-relationships, algebraic operators, and operand relations. Operation descriptors contain either attribute lists or qualifications, which further define the operations at each node. Relation tables contain dictionary information about the relations which are referenced in the query tree, which makes it possible to perform the translation without accessing the target subsystem.

Tree Descriptors

These records define a binary tree which determines the order of operations to be performed. Each record consists of the following fields: operation, left-son, right-son, left-relation, and right-relation. Operation is one of the algebraic operators, left-son and right-son are the sequence numbers of the tree descriptor records which describe that node's left and right successor nodes, and left-relation and right-relation are the names of the relations used at that node.

The third example from Chapter IV, whose query tree is shown again in Figure 61, could be represented by the tree descriptor records which appear in Table 1. The information shown in parenthesis is only explanatory and would not actually appear in the query packet. The hyphenated entries are used to denote that a given node does not have a left and/or right successor or that intermediate results are used as operands.

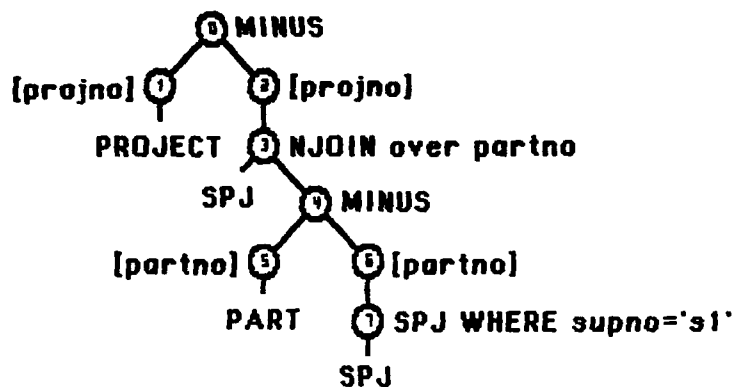


Figure 61: Query tree for Example 3.

Table 1. Tree descriptors for Example 3.

(Rec #)	(Oper)	(L-Son)	(R-Son)	(L-Rel)	(R-Rel)
(0)	MINUS	1	2	-	-
(1)	PROJ	-	-	PROJECT	-
(2)	SEL	3	-	-	-
(3)	NJOIN	-	4	SPJ	-
(4)	MINUS	5	6	-	-
(5)	MINUS	-	-	PART	-
(6)	PROJ	7	-	-	-
(7)	SEL	-	-	SPJ	-

Operation Descriptors

These records are defined according to the algebraic operations that they specify. There are three types of field descriptors: attribute lists, predicates, and relation tables.

Attribute lists. Attribute lists are used to specify attribute names for the projection operation. They consist of the following fields:

N, attr-name1, attr-name2, . . . attr-nameN

where N is the number of attributes in the list and attr-name1 through attr-nameN are the qualified attribute names. Using qualified attribute names helps the translator attach the appropriate tuple variable to each attribute name by replacing each relation name prefix with the proper tuple variable prefix. It also simplifies attribute name aliasing and eliminated ambiguity in projection operations which permute attributes.

Predicates. Predicates are used with selection and join operations. Each predicate consists of a QUEL-compatible qualification string. In order to a predicate to be QUEL-compatible it must use qualified names and all the QUEL comparison operator symbols. Using QUEL-compatible qualification strings saves the translator the task of parsing these predicates. At run time, the translator merely scans the predicate, replaces each relation name prefix with the proper tuple variable prefix, and inserts the qualification string into the QUEL sequence.

Relation tables. Relation tables contain all the attribute names for each relation referenced in the query tree. In this case, however, the attribute names need not be not qualified, that is, they do not contain relation name prefixes since it is understood that all the attributes belong to the relation specified in that record. Relation tables consist of the following fields:

relation-name, N, attr-name1, attr-name2, . . . attr-nameN
where relation-name is self explanatory, N is the number of attributes, and attribute-name1 through attribute-nameN are all the attribute names for that relation, in the order they appear in the view or base relation of the target subsystem. For economy reasons, only one relation table is required for each relation referenced in the query tree.

To avoid ambiguity, delimiters must be used with all three types of descriptors. The operation descriptors for Example 3 are shown in Table 2.

Tree Processing

The objectives of this part of the implementation are to arrive at the correct node ordering for processing and to minimize the number of nodes to be translated.

As mentioned in Chapter IV, the correct order of execution is computed using the sibling pointers for each node and the saved on a stack. The content of this stack can be generated with a recursive function which traverses the tree

Table 2. Operation descriptors for Example 3.

(Record Content)	(Remarks)
1 PROJECT.projno	attr list for node 1
1 SPJ.projno	attr list for node 2
SPJ.parno = PART.parno	join condition for node 3
1 PART.parno	attr list for node 5
1 SPJ.parno	attr list for node 6
SPJ.supno = "S1"	select condition for node 7
SPJ 4 supno parno projno qty	relation table for SPJ
PROJECT 3 projno jname city	relation table for PROJECT
PART 5 parno pname color weight city	relation table for PART

to find all the leaf nodes. This stack is then used to generate the correct order of QUEL statements.

Minimization considerations are made while analyzing the tree in order to produce an optimal number of correctly ordered statements. The criteria for compressing the original tree are actually described in the next section. At this point it is sufficient to note that the stack which defines the correct order of execution for the original tree can be modified after the compressed tree is defined. An alternate method is to integrate the compression criteria into the recursive function so that only the modes containing binary operators are placed on the stack. In either

case, all the information pertaining to unary nodes must be linked to the corresponding compressed nodes.

Operation Mapping

There are two important mechanisms needed to support the translation process: statement generation and attribute tracking. The first one is used to build QUEL statements in a buffer through template modification. The second is used to keep track of the attribute names of relations throughout the translation process. Both mechanisms are discussed below.

Statement Generation

The translation of a node into QUEL takes place via modification of the corresponding template in the work buffer. Modifying the template to reflect the characteristics of the operation specified at that node, is accomplished through following steps: a) insertion of the sequence numbers for intermediate results (TEMPx relations), b) insertion of all pertinent attribute names with proper range variable prefixes into the corresponding target lists, c) replacement of qualified attribute names with appropriate row markers and attribute names in logical predicates and subsequent insertion into the corresponding qualification fields, and d) replacement of attribute names by aliased names generated in preceding intermediate results.

Attribute Tracking

Since QUEL does not permit the use of duplicate attribute names within the same relation, it is necessary to "alias" the names of duplicate attribute names in intermediate results. In order to replace any attribute name by the correct alias name, the translator must keep track of all attribute names. This can be accomplished with an attribute name table with an entry for each attribute name from a temporary relation which contains its relation name and the original qualified name. By tracking attribute names in this manner, the translator is able to look up all attribute names before insert them into the work buffer. The sample attribute name table shown in Table 3 corresponds to the second example presented in Chapter V.

Minimization

Minimization is initiated by identifying the nodes that contain binary operations. These nodes will be the only nodes which will exist after compression. There is one exception: any series of unary operations which are near the root of the tree and which are not followed by any binary operations will also become a single node. Figure 62a identifies the nodes which will exist after compression and Figure 62b shows the resulting tree following compression.

BIOGRAPHICAL SKETCH

Dennis F. Blumenthal was born in Cali, Colombia, in 1950 and came to the United States in 1967. He began his undergraduate studies at Queens College in New York City and later received his Bachelor of Science in Computer Engineering from Syracuse University in 1978. Before coming to the University of Florida, he worked as a programmer for the U.S. Air Force at the Rome Air Development Center in Rome, New York. He also did system level programming for the Directorate of Computer Sciences of the Armament Division at Eglin AFB, Fort Walton Beach, Florida. After completing his graduate studies, he will resume his duties as Computer Systems Officer for the Air Force Technical Applications Center at Patrick AFB, Cocoa Beach, Florida. He currently holds the grade of Captain, is married, and has one son.

Woo81 J. Woodfill, Polly Siegal, Jeff Ranstrom, Marc Meyer, and Eric Allman, INGRES Version 7 Reference Manual (ERL Technical Memo. M79/43), University of California, Berkeley, 1981.

- Mac82 C.R. Maclean, H.M. Bloom, and T.H. Hopp, "The Virtual Manufacturing Cell," presented at the Fourth IFAC Symposium on Information Control Problems in Manufacturing, Gaithersburg, Maryland, October 1982.
- Mac83 C.R. Maclean, M. Mitchell, and E. Barkmeyer, "A Computer Architecture for Small-Batch Manufacturing," in IEEE Spectrum, Vol. 20, No. 5, May 1983, 59-64.
- Mcd83 N. McDonald and J. McNally, "Feature Analysis of INGRES," in Relational Database Systems: Analysis and Comparison, J.W. Schmidt and M.L. Brodie, Editors, Springer-Verlag, New York, 1983.
- NBS85 National Bureau of Standards, "A Distributed Data Management Architecture for Computer Integrated Manufacturing" (Technical Report in preparation), National Bureau of Standards, Washington, D.C., 1985.
- RTI84 Relational Technology Inc., "Graphic Representation of QUEL Syntax" (courtesy poster), Relational Technology Inc., Berkeley, California, 1984.
- Sim82 J.A. Simpson, R.J. Hocken, and J.S. Albus, "The Automated Manufacturing Research Facility of the National Bureau of Standards," in Journal of Manufacturing Systems, Vol. 1 No. 1, 1982, 17-31.
- Smi75 J.M. Smith and P.Y.T. Chang, "Optimizing the Performance of a Relational Algebra Database Interface," in Communications of ACM, Vol. 18, No. 10, October 1975, 568-579.
- Sto76 M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The Design and Implementation of INGRES," in ACM Transactions on Database Systems, Vol. 1, No. 3, 1976, 189-222.
- Su81 S.Y.W. Su, H. Lam, and D.H. Lo, "Transformation of Data Traversals and Operations in Application Programs to Account for Semantic Changes of Databases," in ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, 255-294.
- U1182 J.D. Ullman, "QUEL: A Tuple Relational Calculus Language," in Principles of Database Systems, Computer Science Press, Rockville, Maryland, 1982, 190-197.
- Won76 E. Wong and K. Youseffi, "Decomposition Strategy for Query Processing," in ACM Transactions on Database Systems, Vol. 1, No. 3, 1976, 223-241.

BIBLIOGRAPHY

- Bee83 W. Beeby, "The Heart of Integration: A Sound Database," in IEEE Spectrum, Vol. 20, No. 5, May 1983, 44-48.
- Cer in S. Ceri and G. Gottlob, "Translating SQL into
press Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries," Politecnico di Milano, to appear in IEEE Transactions on Software Engineering.
- Chu82 W.W. Chu and P. Hurley, "Optimal Query Processing for Distributed Database Systems," in IEEE Transactions on Computers, Vol. 1, No. 3, September 1982, 835-850.
- Cod72 E.F. Codd, "Relational Completeness of Data Base Sublanguages," in Data Base Systems, R. Rustin, Editor, Prentice-Hall, New York, 1972.
- Dat82 C.J. Date, An Introduction to Database Systems, Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.
- Day81 U. Dayal, N. Goodman, and R. Katz, An Extended Algebra with Control over Duplicate Elimination, Computer Corporation of America, Boston, 1981.
- Hel75 G.D. Held, M.R. Stonebraker, and E. Wong, "INGRES-A Relational Data Base System," in AFIPS, Vol. 44, 1975, 409-416.
- Hop83 T.H. Hopp and K.C. Lau, "A Hierarchical, Model-Based, Control System for Inspection," presented at First ASTM International Symposium on Automated Manufacturing, San Diego, California, April 1983.
- IEE83 IEEE, "Data Driven Automation," in IEEE Spectrum (compendium of selected articles), Vol. 20, No. 5, May 1983, 34-96.
- Klu80 A. Klug, "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions" (Technical Report 389), Computer Science Department, University of Wisconsin, Madison, Wisconsin, June 1980.

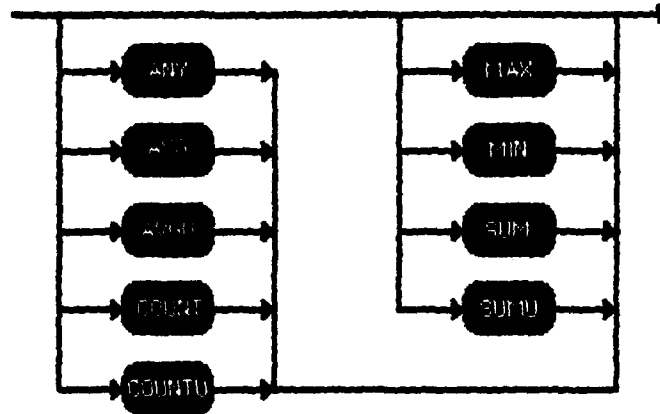


Figure 78: Aggregate Name specification.

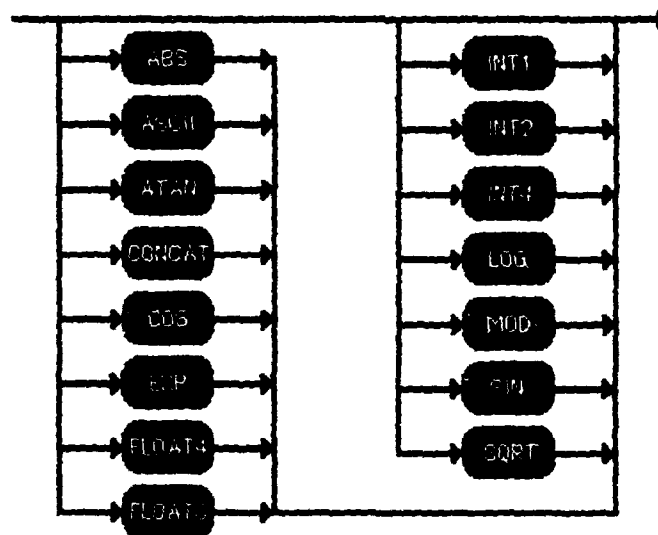


Figure 79: Function Name specification.

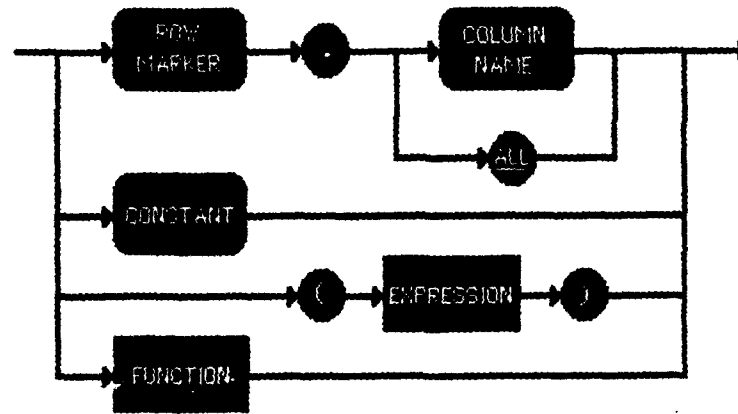


Figure 75: Primary specification.

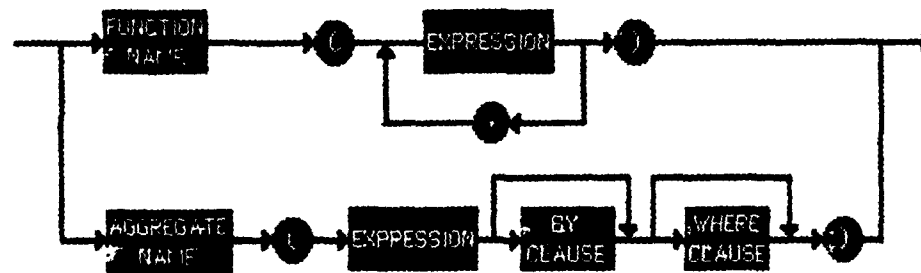


Figure 76: Function specification.

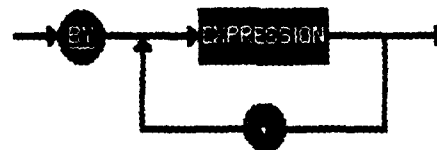


Figure 77: By Clause.

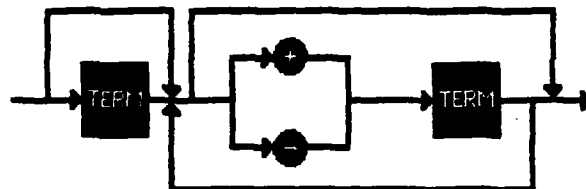


Figure 72: Expression specification.

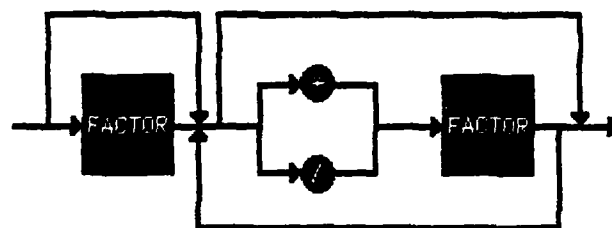


Figure 73. Term specification.

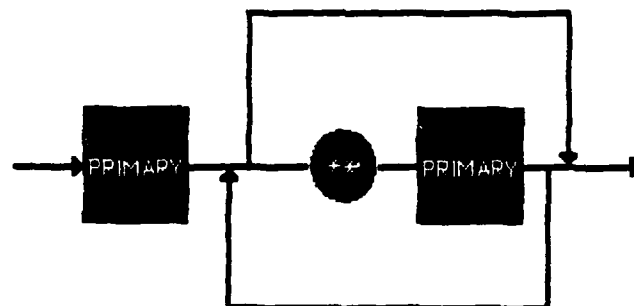


Figure 74: Factor specification.

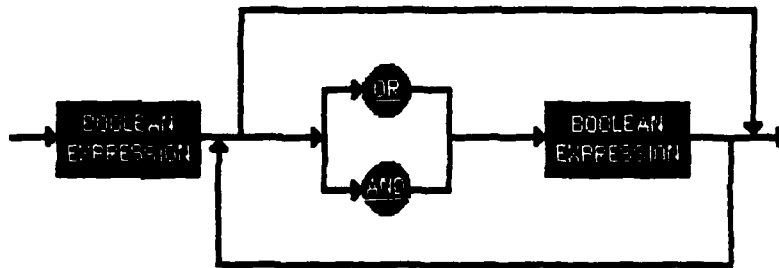


Figure 69: Qualification specification.

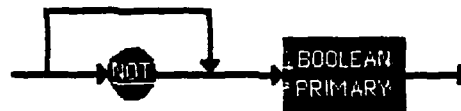


Figure 70: Boolean expression specification.

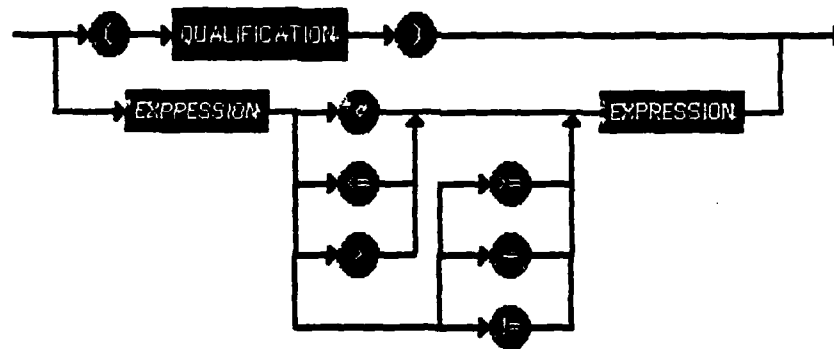


Figure 71: Boolean primary specification.



Figure 64: The APPEND statement.



Figure 65: The DELETE statement.

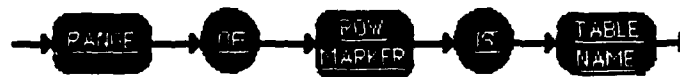


Figure 66: The RANGE statement.

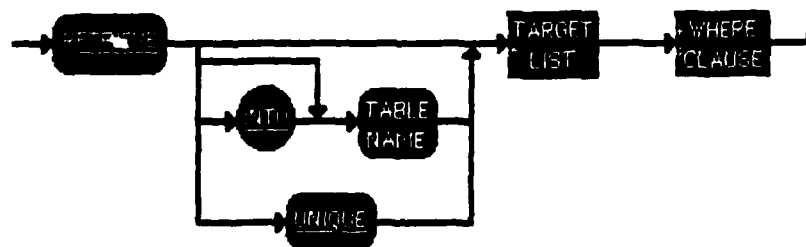


Figure 67: The RETRIEVE statement.



Figure 68: The WHERE clause.

APPENDIX INGRES SYNTAX

The purpose of this appendix is to acquaint the reader with the portions of the INGRES syntax which are used reoccurringly in the chapters of this thesis. The syntax is presented in graphic form. The graphic constructs used are shown in Figure 63. Figures 63a and 63b represent atomic constructs, i.e. they cannot be recursively decomposed. Figure 63c represents a construct which is not atomic, i.e. it can be made up of atomic or non-atomic constructs.

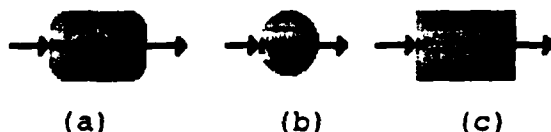


Figure 63: Graphical syntax constructs.

The words which appear within the graphic constructs have two forms: keywords and non-keywords. Keywords are always underlined, non-keywords are not. The INGRES statements which are shown are: APPEND (Figure 64), DELETE (Figure 65), RANGE (Figure 66), and RETRIEVE (Figure 67). In addition, the WHERE clause (Figure 68) and its components (Figures 69-79) are also shown.

It is recommended that an extended version of the relational algebra be developed to better utilize the processing capabilities of other target subsystems.

Another issue addressed by this author is the scope of the patterns used for the minimization rules presented in Chapter V. The translation algorithm presented in Chapter IV obviously generates too many QUEL statements and the algorithm presented in Chapter V is a worthwhile improvement. However, it is debatable whether the latter is truly optimal, e.g. there are more compact translations for the query examples presented in Chapter V. In considering pattern scope, this author contends that a generalized algorithm with patterns of wider scope would be too complex and would not be reusable in other translators, mainly because the minimization technique depends on the target DML's semantic capabilities. Thus, the author formulated a "middle-of-the-road" solution, embodied in the Type II pattern. In any case, the search for a truly optimal translator merits further study.

To others undertaking future work on similar translators, this author recommends that analytical studies be made about the relationships between the relational algebra and the underlying semantics of the target DML. Thinking in retrospect, mapping those relationships would have been helpful to this author, e.g. converting relational algebra to tuple calculus before translating into QUEL.

CHAPTER VII CONCLUSION

An algorithm to translate algebraic queries to QUEL was presented in the previous chapters. At the time of this writing, implementations of the node-per-node algorithm and the optimal algorithm have been tested successfully. This author is compelled to point out that, while this work resulted in a practical and feasible solution, it is by no means unique and other alternatives became evident during the course of the work.

One of the issues confronted by the author deals with the limited expressive power of the relational algebra. QUEL has a great deal more expressive power, e.g. aggregate functions, mathematical functions, sorting functions, etc., not found in the relational algebra. Consequently, a large portion of the processing power of the INGRES subsystem remains unexploited. This author made some allowances to compensate for this, namely, a non-parsing technique is used to build the predicates for the QUEL translation. This technique is flexible enough to allow the usage of aggregate mathematical, and sorting functions in QUEL predicates and make better use of the INGRES subsystem's processing capabilities.

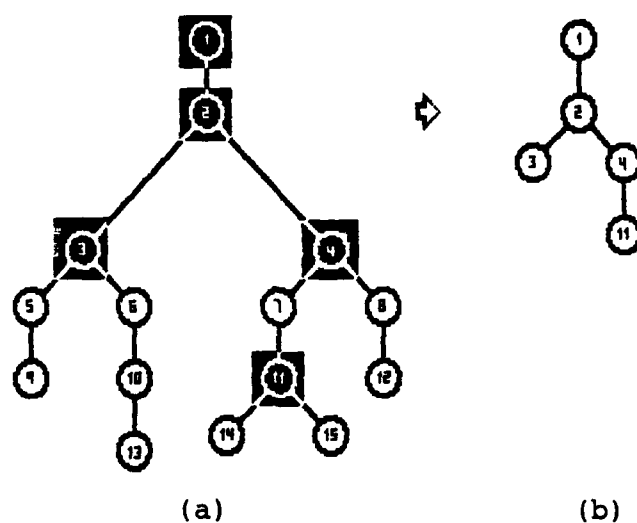


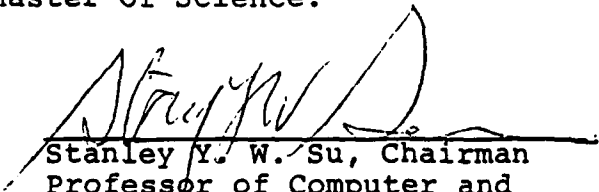
Figure 62: Query tree compression.

When compression takes place, all operations in the query tree must be preserved. The operations specified in the nodes that are deleted must be integrated into remaining nodes according to the rules presented in Chapter V. The six templates previously presented, combine several unary operations with one binary operation. A compressed node has all the operations which were originally at that node plus the left and right topmost attribute lists and composite predicates which are extracted from nodes being deleted.


Table 3. Attribute name table for Example 2 (optimized).

(attrnam)	(relation)	(origin)
supno	TEMP1	SUPPLIER.supno
city	TEMP1	SUPPLIER.city
supno2	TEMP1	SPJ.supno
partno	TEMP1	SPJ.partno
projno	TEMP1	SPJ.projno
qty	TEMP1	SPJ.qty
supno	TEMP2	TEMP1.supno
city	TEMP2	TEMP1.city
supno2	TEMP2	TEMP1.supno2
partno	TEMP2	TEMP1.partno
projno	TEMP2	TEMP1.projno
qty	TEMP2	TEMP1.qty
projno2	TEMP2	PROJECT.projno
city2	TEMP2	PROJECT.city
city	TEMP3	TEMP2.city
partno	TEMP3	TEMP2.partno
city2	TEMP3	TEMP2.city2


I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.


Stanley Y. W. Su, Chairman
Professor of Computer and
Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.


Charles V. Shaffer
Professor of Electrical
Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.


Herman Lam
Assistant Professor of
Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Michael V. Mannino
Assistant Professor of
Computer and Information
Sciences

This thesis was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Master of Science.

May 1985

Dean, College of Engineering

Dean, Graduate School

END

FILMED

9-85

DTIC